Trabalho Nº 1 - Linux e C

1. Introdução

O sistema Linux e a linguagem C são as ferramentas essenciais de aprendizagem na cadeira de Sistemas Operativos.

O sistema Linux é hoje em dia utilizado por milhões de pessoas e empresas em todo o mundo e constitui uma alternativa séria e robusta aos sistemas proprietários comerciais. Trata-se de um sistema avançado e completo, que além disso está em constante evolução. Uma das suas grandes vantagens é ser um sistema de código aberto (Freeware), ao qual qualquer um pode aceder.

Por este motivo este sistema constitui uma excelente ferramenta pedagógica, pois a leitura e interpretação do seu código fonte permite compreender, através de exemplos reais, a forma como podem ser implementadas as funcionalidades que constituem um Sistema Operativo.

Uma vez que o sistema Linux está essencialmente escrito em C, na maior parte dos casos utilizando os seus conceitos mais avançados, torna-se essencial adquirir rapidamente excelentes conhecimentos nessa linguagem.

Com este trabalho pretende-se que os alunos adquiram ou revejam esses conhecimentos, através da realização de programas que irão utilizar directamente funcionalidades do sistema operativo.

2. Ambiente de Trabalho

Cada aluno tem acesso a uma estação de trabalho individual, no qual está instalado o sistema Linux Fedora Core 7.

Neste primeiro trabalho irá familiarizar-se com o ambiente gráfico (GDM) e os principais comandos do sistema, assim como com o ambiente de desenvolvimento que permite a edição, compilação e execução de programas.

2.1 Criação de Utilizador

Cada aluno deverá escolher uma estação de trabalho individual que em princípio irá utilizar durante todas as aulas práticas da cadeira. Nessa estação deverá criar um utilizador que terá por login o seu identificador de aluno, ou seja:

```
a1234567
```

Não deverão ser utilizados nomes próprios como identificadores de login. Para a criação do utilizador, deverá contactar o professor.

2.2 Pasta de trabalho

Uma vez criado o utilizador, este irá dispor de uma pasta ou directoria de trabalho que estará situada em

```
/home/a1234567
```

É aconselhado criar aí uma pasta por cada trabalho a realizar. Assim, o trabalho 1 irá ser realizado no directório:

```
/home/a1234567/trabalho1
```

É aconselhável não utilizar espaços nos nomes de pastas ou ficheiros pois o interpretador de comandos (**bash**) não gere os nomes com espaços. Por outro lado maiúsculas e minúsculas são interpretados diferentemente (é *case sensitive*).

2.3 Interpretador de comandos

Depois de realizar o login no sistema, o utilizador dispõe de um interpretador de comandos que lhe permite executar as tarefas pretendidas. Em Linux esse interpretador chama-se **bash** e deverá adquirir alguma experiência da sua utilização.

Cada vez que se escreve um comando e se carrega em *Enter*, o bash executa o comando indicado, apresenta os resultados e volta a imprimir um prompt (**$**) que indica que está de novo pronto a receber um comando.

Exemplos:

```
$ pwd
/a123456/trabalho1
$ date
Sun Oct 1 02:08:45 WEST 2006
$ ls
Desktop Downloads Examples hello hello.c
$ mkdir trabalho1
$ ls -ld trabalho1
drwxr-xr-x 2 a1234567 users 4096 2006-10-01 02:11 trabalho1
```

3. Ambiente de desenvolvimento C

Para escrever programas e os executar, é necessário utilizar um conjunto de ferramentas que no seu conjunto constituem um ambiente de desenvolvimento.

O ambiente de desenvolvimento C em Linux pode ser extremamente simples, sendo no mínimo constituído pelas aplicações que a seguir se descrevem.

3.1 Editor de Texto

É a aplicação que irá permitir a escrita do programa. Inúmeros editores de texto estão disponíveis em Linux, desde os mais simples (**vi** e **jed**) até aos mais complexos (**emacs**) passando pelos mais clássicos **kate** e **kwrite**, que fazem parte das aplicações disponíveis na barra de comandos do ambiente gráfico.

Os ficheiros criados contendo programas em C deverão ser guardados com a extensão .c Assim, por exemplo para editar um programa hello.c com o editor kate pode-se escrever:

```
$ kate hello.c
```

Ou então abrir o editor através do menu de programas e procurar o programa na pasta de trabalho. Se este não existir deverá obviamente ser usada a opção **Save as** para o guardar.

3.2 Compilador C

Uma vez criado o programa em C, é necessário transformá-lo num programa executável. Para isso é utilizado o compilador C (**cc** ou **gcc**) que irá realizar uma série de operações sobre o programa (pré-processamento, verificação do código, compilação, assemblagem, edição de ligações), para finalmente criar a imagem executável.

Para realizar esta operação deve ser executado o comando:

```
$ cc hello.c
```

Se não for especificado o nome do ficheiro executável, este recebe o nome por defeito que é **a.out.**

Para definir o nome do programa executável utiliza-se a opção:

```
$ cc hello.c –o hello
```

3.3 Execução e Debug

A execução de um programa é feita através da invocação do seu nome na linha de comandos. Assim para executar o programa acima criado:

```
$ ./hello
```

A utilização do “**./**“ em frente do nome é necessária para que o **bash** procure o nome na pasta em que foi criado o executável.

Existe outra forma de definir a pasta onde se encontram os programas executáveis do utilizador, através da modificação da variável de ambiente (*environment*) **PATH**, que indica ao interpretador de comandos onde deve procurar os comandos executáveis. Por exemplo o comando:

```
$ PATH=$PATH:/home/a1234567/trabalho1
```

adiciona à variável **PATH** o caminho para a pasta **trabalho1** onde estão armazenados os ficheiros deste trabalho.

No caso do programa apresentar erros na sua execução que sejam difíceis de encontrar, pode-se recorrer ao GNU Debugger, uma aplicação que permite realizar a execução do programa de forma controlada.

Para tal é necessário compilar o programa com a opção -g de forma a que o executável guarde infomação adicional sobre o programa (tabela de símbolos, etc…):

```
$ cc hello.c –g –o hello
```

O *debugger* é de seguida invocado através do comando:

```
$ gdb hello
```

Desta forma o *debugger* toma o controlo da sessão, ou seja os comandos inseridos serão agora por ele interpretados, e não pelo bash. Assim, o *prompt* que aparece agora é o seguinte:

```
(gdb)
```

As opções desta ferramenta são múltiplas, e convém consultar o manual de utilização, através da invocação do comando **`man gdb`** (ou consultando o manual on-line).

As opções mais utilizadas são as seguintes:

- p: para listar o código fonte do ficheiro
- b: para colocar um *breakpoint* (ponto de paragem)
- r: para lançar a execução do programa
- s: para executar o programa passo a passo

Exemplo:

```
(gdb) l
1       main()
2       {
3               printf("Hello\n");
4       }
(gdb) b 3
Breakpoint 1 at 0x804837c: file hello.c, line 3.
(gdb) r
Starting program: /home/jrogado/programs/hello

Breakpoint 1, main () at hello.c:3
3               printf("Hello\n");
(gdb) s
Hello
4       }
```

Vemos no exemplo a listagem, inserção de um *breakpoint*, execução do programa que pára no *breakpoint* inserido, e a execução passo a passo da instrução printf. O programa poderia ser assim executado passo a passo até se descobrir a eventual causa de erro.

4. Trabalho a realizar

Depois de se ter familiarizado com o ambiente de trabalho, irá agora abordar um trabalho de programação em C.

O programa seguinte realiza uma série de operações sobre *strings* (cadeias de caracteres) utilizando conjunto de métodos ou funções auxiliares das quais só são fornecidas as definições.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#define BUFSIZE 512

int string_size(char *str);
int string_compare(char *s1, char *s2);
char *string_copy(char *dest, char *source);
char *string_append(char *dest, char *source);

main(int argc, char *argv[])
{
   char *str1, *str2;
   int size1, size2;
   char buffer[BUFSIZE];
   int equal, size;

   if (argc != 3){
```

```
        fprintf(stderr, "Usage %s <string1> <string2>\n", argv[0]);
        _exit(1);
  }
  str1 = argv[1];
  str2 = argv[2];

  // Calcular o comprimento de cada string e imprimir os valores
  size1 = string_size(str1);
  size2 = string_size(str2);
  printf("String 1: %s %d String 2: %s %d\n", str1, size1, str2, size2);

  // Comparar as duas strings e indicar se sao iguais ou diferentes
  equal = string_compare(str1, str2);

  // Copiar a primeira string para um novo buffer
  // testando primeiro se ha espaco para a copia
  string_copy(buffer, str1);

  // Juntar a segunda string ao final primeira da
  string_append(buffer, str2);

  size = string_size(buffer);

  // Imprimir a string resultante e o seu tamanho
  // testando primeiro se ha espaco para a copia
  printf("String resultante: %s %d\n", buffer, size);
}
```

4.1 Copie o código acima para um ficheiro com o nome **`strings.c`** e experimente realizar a sua compilação. Interprete os erros que obtém.

4.2 Implemente as funções que faltam de forma a não obter erros de compilação e a que o programa implemente as funcionalidades esperadas. Não se esqueça de realizar os testes de coerência de forma a evitar ultrapassar o tamanho do **`buffer`** utilizado para a cópia.

4.3 Substitua as funções que implementou por funções da biblioteca C de forma a que as funcionalidades se mantenham.

5. Prazo de Entrega do Trabalho

O prazo para a entrega dos trabalhos é de duas semanas a contar da aula em que o enunciado foi apresentado. Não serão aceites trabalhos fora do prazo. A entrega deverá ser feita por e-mail num ficheiro zip (ou rar) contendo um relatório em PDF e as listagens dos programas realizados, obedecendo OBRIGATORIAMENTE ao seguinte formato:

`a123456-trabalho-N.zip`

6. Referências

"The C Programming Language, 2nd Edition (Ansi C)" de by Brian W. Kernighan, Dennis M. Ritchie, ed. Prentice Hall.

"*The C Book,* second edition, by Mike Banahan, Declan Brady and Mark Doran, ed. Addison Wesley http://publications.gbdirect.co.uk/c_book

Manual de Referência (man pages) Linux on-line: http://man.he.net